Floating-Point Cleanup

WG14/N757 J11/97-120 (Draft 1997-09-26)

Jim Thomas
jthomas@best.com

This documents describes two enhancements to the floating-point specification. Section numbers refer to C9X Draft 11-pre3.

1. Enhancement: Align the **`scalb`** function to X/Open and make one other minor change to **`scalb`** for consistency. There are four points:

a) Currently C9X specifies that **`scalb(x, y)`** returns a NaN if **`y`** is finite and nonintegral. For X/Open alignment C9X should specify that a finite nonintegral second argument is first truncated to an integral value. See rationale below.

b) X/Open specifies "On implementations that support IEEE, *scalb()* returns *x* when *x* is infinity", implying **`scalb(`**+/-Inf**`,`** -Inf**`)`** must be +/-Inf. However, from basic principles, it should be NaN, as +/-Inf * 2^(-Inf) = +/-Inf * 0 = NaN. (If any non NaN result were considered it should be +/-0, not +/-Inf, because x * 2^y is zero for large-magnitude x and large-magnitude negative y.) C9X should retain its current specification and officially recommend X/Open change its specification.

c) For the case **`scalb(`**+Inf**`,`** NaN**`)`**, X/Open contradicts itself in saying "On implementations that support IEEE, *scalb()* returns *x* when *x* is infinity" but also "On implementations that support IEEE NaN, if an input argument is NaN, then the function returns NaN." C9X should retain its current specification for a NaN result in this case, and officially recommend X/Open clarify it specification to concur.

d) C9X should specify in Appendix F that **`scalb(`**+∞**`,`** -∞**`)`** raise the invalid exception.

Rationale: Regarding (a), the discrepancy arose from a misunderstanding about the X/Open specification when the second argument is finite and nonintegral. Presuming X/Open had left these cases unspecified, C9X specified NaN results for them, because the IEEE 754 committee, whose standard recommends a scalb function, didn't intend the function for nonintegral finite second arguments, and because Sun's fdlibm **`scalb`** returns NaN for finite nonintegral second arguments. However, X/Open does specify that "On implementations that support IEEE, *scalb()* returns *x* when *x* is infinity." And validation suites test cases where the first argument is infinite and the second is finite nonintegral (though they don't to my knowledge test finite nonintegral second arguments generally).

X/Open also says that **`scalb`** is equivalent to **`ldexp`**. Although this statement is incomplete, as **`ldexp`**'s second argument has type **`int`**, it would seem to imply that the second argument should be truncated (like assignment) to an integer.

This proposal resolves the discrepancy by changing C9X to specify that a finite nonintegral second argument to **`scalb`** is first truncated to an integral value. For example, with this change, **`scalb(1.0, 3.987)`** would return 8.0 instead of a NaN.

Regarding (d), this change makes **scalb** consistent with the general treatment of NaNs generated from non NaN arguments.

Edits:

Page 211, 7.7.6.12 The **scalb** function
Change the second sentence in the Description to "If **y** is finite and nonintegral, it is first truncated to an integral value."

Page 457, F.9.3.12 The **scalb** function
Change the bullets to

- **scalb(-x, y)** and **-scalb(x, y)** are equivalent.
- **scalb(x,** ±0**)** returns **x**.
- **scalb(**+0**,** +∞**)** returns a NaN and raises the invalid exception.
- **scalb(**+0**, y)** returns +0, for **y** < +∞.
- **scalb(**+∞**, y)** returns +∞, for **y** > -∞.
- **scalb(x,** -∞**)** returns +0, for finite positive-signed **x**.
- **scalb(x,** +∞**)** returns +∞, for positive **x**.
- **scalb(**+∞**,** -∞**)** returns a NaN and raises the invalid exception.
- **scalb(x, y)** returns a NaN, if **x** or **y** is a NaN.
- Whether **scalb(x, y)** raises the inexact exception when **y** if finite and nonintegral is unspecified.

2. Enhancement: Loosen the restrictions on the definitions of the exception macros in **<fenv.h>**. Currently they are required to be powers of 2. We should allow them to be any values such that all combinations of bit-wise ORs are unique.

Rationale: Having the bit-wise ORs be unique was the only purpose for defining them to be powers of 2. The proposed more liberal specification would allow the implementation to include system id bits in the values, which could facilitate supporting binary compatibility.

Edits:

Page 188, 7.6 Floating-point environment **<fenv.h>**
In [5] change "macros expand to integral constant expressions whose values are distinct powers of 2" to "macros expand to integral constant expressions with values such that bitwise ORs of all combinations of the macros are unique".